Interfacing the Am186™CC Communications Controller to an AMD SLAC™ Device Using Enhanced SSI

AMD

Application Note

by Jeff Kirk

The purpose of this application note is to describe how to interface an Am186™CC communications controller to a Quad Subscriber Line Audio-Processing Circuit (QSLAC™) device. The same techniques can be used to interface an Am186CC controller to any SLAC™ device.

BACKGROUND

Traditionally, linecards—if they had processors at all—used simple, inexpensive 8-bit microcontrollers. However, as the number of lines per card increases,16-bit controllers such as the Am186CC controller become more attractive for several reasons:

- 16-bit controller costs have decreased
- More peripheral functions can be integrated, reducing external component counts
- Smaller packaging options are now available
- 16-bit controllers generally offer larger address spaces

These factors, combined with the availability of superior, low-cost development tools such as the Microsoft® and Borland C compilers, reduce the time to market and long-term maintenance costs.

The Am186CC controller also offers several new features useful in communication systems:

- Mixed 8- and 16-bit memory/peripheral spaces
- Programmable I/Os with dedicated set/clear functions
- High-speed High-level Data Link Control (HDLC) ports (10 Mbit/s)
- Direct Pulse Code Modulation (PCM) Highway interfaces

PCM interfaces—up to four each—provide simple processor access to data on the SLAC device's PCM interface. Also, one of the Am186CC controller's HDLC ports, Channel A, supports General Circuit Interface (GCI) operation. GCI is also known as the ISDN Oriented Modular Interface, Revision 2 (IOM-2).

A SLAC device connects to its host processor through a three-pin serial interface. While this interface is used primarily to initialize the SLAC device, several critical functions of the Subscriber Line Interface Circuit (SLIC) can be monitored through the serial interface. As a result, it may be necessary to make the interface as fast as possible. Designed to drive the SLAC device's Microprocessor Interface (MPI) bus, the SSI port of the Am186CC controller makes it even easier for the hardware and software engineer to implement the fastest possible interface.

FURTHER REFERENCES

The remainder of this application note assumes at least a passing familiarity with the chips involved: the Am186CC controller and the Am79Q02 QSLAC device. If further details are needed, the following literature is available from AMD:

Am186™CC Communications Controller Data Sheet, order #21915

Am186™CC Communications Controller User's Manual, order #21914

Am186™CC Communications Controller Register Set Manual, order #21916

Am186™ and Am188™ Family Instruction Set Manual, order #21267

Am79Q02/021/031 QSLAC™ Data Sheet, order #18503

AMD's complete family of linecard devices is found in the *Linecard Products for the Public Infrastructure Market Data Book,* order #18503.

Publication# **21921** Rev: **A** Amendment/**0**
Issue Date: **June 1998**

MPI HARDWARE OVERVIEW

The MPI buses of the QSLAC device and the DSLAC device are very similar. Both are serial, master/slave-type interfaces. A system or linecard microprocessor is the master, and the interface is designed so that multiple slaves (i.e., SLAC devices) can be attached to a single master's MPI bus, as shown in Figure 1.

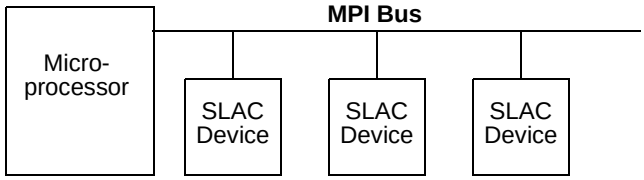

Figure 1. Multiple SLAC Devices

The MPI bus signals, like those through most digital buses, are of three types:

- Data
- Clock/control
- Address

The QSLAC data line (DIO) pin is a bidirectional, three-state serial bus. Some DSLAC devices, like the Am79C02 DSLAC device, has separate Data In (DIN) and Data Out (DOUT) pins, which can be strapped together to look like the QSLAC device's single DIO pin. The data on this line consists of 8-bit bytes transmitted most significant bit first, regardless of direction. The master initiates all transfers by sending a command byte to the SLAC device. Each command has a predetermined length (number of bytes) and direction (read or write). For example, if the master microprocessor sends out command number 25 (read GX filter coefficients) to the DSLAC device, the DSLAC device knows to transmit two bytes. Because the command determines which device is transmitting, master or slave, the software drivers must be correct to prevent bus contention, which could damage the devices. Also, in the case of a read, the SLAC device will not accept a new command until the old one is finished (i.e., until all bytes have been received or transmitted). Software verification is critical.

The clock signal (DCLK) is an input to the SLAC device. The clock can run continuously or can be active only during data transfers. The maximum frequency of DCLK for both QSLAC and DSLAC devices is 4.096 MHz. Data is clocked into the SLAC device on the rising edge of DCLK, but data is sent out on the falling edge of DCLK. This common technique makes it easier to satisfy setup and hold-time requirements. DCLK can be stopped indefinitely in either the High or Low state if the chip select input is held High.

Each of the individual SLAC devices on the MPI bus is addressed (i.e., selected) by pulling one of the chip select inputs Low. The QSLAC device has a single chip select ($\overline{CS}$) for all four channels, while the DSLAC device has a separate chip select for each channel ($\overline{CS1}$ and $\overline{CS2}$). The rising edge of the chip select marks or frames the end of each byte; therefore, the chip select line must go High for at least the minimum off period before the next byte is read or written. The DSLAC device's minimum off period is 5 µs, while the QSLAC device's minimum is 2.5 µs.

If the QSLAC device receives 16 clocks with $\overline{CS}$ asserted (Low), then the device resets.

In addition to the data, clock, and address pins, the QSLAC device has an interrupt pin as part of the microprocessor interface. This pin can be very useful in some systems, but is not discussed in this application note. A brief description of this pin is available in the *Am79Q02/021/031 QSLAC™ Data Sheet,* order #18503.

SSI HARDWARE OVERVIEW

The Enhanced Synchronous Serial Interface (SSI) of the Am186CC controller was designed to interface directly with AMD SLAC devices and to provide a low-pin-count interface with application-specific integrated circuits (ASICs). With the right clocks, the Am186CC controller can drive the MPI bus at its maximum rate.

The SSI bus transmits three signals, each on a separate pin:

- SDATA
- SCLK
- SDEN

All the pins are shared (i.e., multiplexed) with one of the Am186CC controller's 48 PIOs. This allows the SSI pins to be used as PIOs if their normal SSI function is not needed. The pins are PIOs by default.

The SDATA signal—like DIO—is a bidirectional, three-state serial bus. Unlike DIO, a weak pullup or pulldown resistor keeps the last value on the bus for systems that cannot tolerate three-state inputs. The data on this signal consist of 8-bit bytes, normally transmitted least significant bit first, but SSI can be programmed for MSB-first operation. The master/slave protocol is controlled entirely with software.

The clock signal (SCLK) is active only during byte transfers. It is an output signal. The frequency is derived by dividing the frequency of the internal clock by 2, 4, 8, 16, 32, 64, 128, or 256 (programmed with the

SSCON register). In the case of a 40-MHz device, this allows for speeds up to 20 MHz. As with the SLAC device, data is clocked out on the falling edge and clocked in on the rising edge.

The enable signal, SDEN, is an output signal. It is normally active High, but it can be programmed to be active Low to match $\overline{CS}$ of the MPI interface. While there is only one enable pin, PIOs can be easily used as chip selects to connect multiple SLAC devices to the SSI port.

CONNECTING SSI TO MPI

The hardware connection is very easy. Figure 2 shows the connections for the QSLAC device.

If interfacing the Am186CC controller with the DSLAC device (which has two chip selects) or with multiple SLAC devices, simply use PIOs in place of SDEN (see Figure 3). There is nothing special about PIO9 and PIO10; any uncommitted PIOs work. If the SLAC device has separate DIN and DOUT pins rather than a DIO pin, simply tie them together.

Am186CC Controller (PQFP)		Am79Q02 QSLAC Device (PLCC)
SDATA (pin 4)	↔	DIO (pin 38)
SCLK (pin 3)	→	DCLK (pin 39)
SDEN (pin 2)	→	$\overline{CS}$ (pin 40)

Figure 2. QSLAC Device Connections

Am186CC Controller (PQFP)		Am79C03 DSLAC Device (PLCC)
SDATA (pin 4)	↔	DIO (pin 21)
SCLK (pin 3)	→	DCLK (pin 19)
PIO10 (pin 2)	→	CS1 (pin 32)
PIO9 (pin 124)	→	CS2 (pin 31)

Figure 3. DSLAC Device Connections

TIMING CONSIDERATIONS

The tables on page 4 compare the timing requirements for the worst cases. Each worst case is determined by looking at the most stringent requirement to see if the other end of the interface can meet the requirement. There is only one speed grade for the QSLAC device, but there are multiple speed grades for the Am186CC controller, so each case has been looked at with the worst possibility in mind.

The timing of the DSLAC device is largely the same as the timing of the QSLAC device. The only difference is that the Chip Select Off time (parameters 9 and 16) is longer—5 µs versus 2.5 µs—on the DSLAC device.

When verifying the timing, there are two cases to consider: microprocessor write cycles and microprocessor read cycles. Table 1 on page 4 looks at the write cycle case, while Table 2 examines read cycles. These numbers are easily verified, but as an example, the next few paragraphs explain how Table 1 was derived. Verifying Table 2 is left as an exercise for the reader.

During a write cycle, the QSLAC device's DIO pin is an input, so it has setup and hold timing requirements as illustrated in Table 1. These requirements can be obtained directly from the QSLAC device's data sheet as parameters number 10 and 11 (Input data setup time, t_{IDS}, and Input data hold time, t_{IDH}). Chip select is also an input, with similar setup and hold times. The QSLAC device specifies these parameters separately for the input (write) and output (read) cases so the correct parameters are 6 and 7 (Chip select setup time, input state t_{ICSS}, and Chip select hold time, input state t_{ICSH}).

Determining what the Am186CC controller provides—the right half of Table 1—is a little more complicated. Remember that SCLK and DCLK are tied together (i.e., they are the same clock.) Because data from the Am186CC controller is output on the falling edge of SCLK and latched into the QSLAC device on the rising edge of DCLK, there is one-half clock cycle between the two events. Assume the worst case duty cycle the QSLAC device can tolerate, which is parameter 3 (Data Clock Low Pulse Width, t_{DCL}) at 97 ns. This is a conservative assumption because SCLK is always very close to 50%. The Am186CC controller data sheet guarantees that SDATA is valid no more than 20 ns after SCLK goes Low in parameter t_{SLDV} (SCLK Low to Data Valid). Subtracting this delay from the length of the clock pulse (97–20=77) leaves 77 ns for setup time before the rising edge of DCLK.

All the other setup and hold times in Table 1 and Table 2 are calculated in a similar fashion, and as the tables show, there are generous margins even in the worst case.

In the case of the data hold time in Table 1, there is no specification given in the Am186CC controller data sheet for how quickly SDATA can change after the clock goes Low. It is assumed that in the worst case, SDATA instantaneously changes as soon as SCLK goes Low. This means that the hold time provided by the Am186CC controller is the same as the minimum SCLK High period, which is specified by the QSLAC device as 97 ns.

The chip-select-setup and hold times are given for SDEN rather than PIOs as discussed previously. Even though SDEN is driven by the SSI, SDEN is still controlled by software by writing a 1 or 0 to the DE bit in the SSI Control (SSCON) register. Because SDEN is controlled by software, the actual delay is much longer than specified. The same holds true if PIOs are used to drive the SLAC device's chip selects.

Table 3 gives the usable options for each of the available speed grades and the resultant data transfer rate. To achieve the maximum DCLK rate of 4 MHz, the Am186CC controller's internal frequency must be 8, 16, or 32 MHz (÷ 2, 4, or 8).

Table 1. Microprocessor Output (Data Write)

Timing Parameter	QSLAC Device Requires (ns, min)	Am186CC Controller Provides (ns, min)	Timing Conflicts
Data Setup time	30	72	None
Data Hold time	30	97	None
CS Setup time	70	219	None
CS Hold time	0	122	None

Table 2. Microprocessor Input (Data Read)

Timing Parameter	Am186CC Controller Requires (ns, min)	QSLAC Device Provides (ns, min)	Timing Conflicts
Data Setup time	10	47	None
Data Hold time	3	97	None

Table 3. Am186CC Controller Frequencies

Clock Divisor	Am186CC Controller -20 MHz	Am186CC Controller -32 MHz	Am186CC Controller -40 MHz	Am186CC Controller -48 MHz	Am186CC Controller -50 MHz
÷2	–	–	–	–	–
÷4	–	–	–	–	–
÷8	2.5 MHz	4 MHz	–	–	–
÷16	1.25 MHz	2 MHz	2.5 MHz	3.0 MHz	3.13 MHz
÷32	625 KHz	1 MHz	1.25 MHz	1.5 MHz	1.56 MHz
÷64	313 KHz	500 KHz	625 KHz	750 KHz	781 KHz
÷128	156 KHz	250 KHz	313 KHz	375 KHz	391 KHz
÷256	78 KHz	125 KHz	156 KHz	187.5 KHz	195 KHz

Note: *These frequencies are not the orderable speed grades, but are frequencies chosen that are a multiple of the fastest MPI clock, which is 4 MHz.*

SOFTWARE CONSIDERATIONS

The basics of using the SSI port from software can be illustrated with two subroutines. The first subroutine writes a byte to the SLAC device; the second reads a single byte. These two routines, along with initialization, form the core of the necessary drivers.

The SSI port appears as five registers in the Am186CC controller's peripheral control block. This 1-Kbyte block can be located either in memory or in I/O space at the location pointed to by the Peripheral Control Block Relocation (RELOC) register. The RELOC register resides in the last register address of the peripheral control block, at offset 03FEh. Because the base location of the block can be moved, the location of individual registers is specified as an offset from the RELOC register rather than as an absolute address. The PIO ports and control registers are also located in this block of addresses. At reset, the peripheral control block is located at FC00h in I/O space. This places the RELOC register at FFFEh.

Table 4 shows the five SSI registers. The bit-level definitions of these registers are given in Table 5.

Table 4. SSI Port Registers

Offset from PCB	Register Mnemonic	Register Name
2F0h	SSSTAT	SSI Mode/Status
2F2h	SSCON	SSI Control
2F4h	SSTXD1	SSI Transmit 1
2F6h	SSTXD0	SSI Transmit 0
2F8h	SSRXD	SSI Receive

Table 5. Bit Level Definitions

Offset	Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2F0h	SSSTAT	ENHCTL													RE/TE	DR/DT	PB
2F2h	SSCON					CLKP	DENP		MSBF		CLKEXP					DE1	DE0
2F4h	SSTXD1									TXDATA							
2F6h	SSTXD0									TXDATA							
2F8h	SSRXD									RXDATA							

The Port Busy (PB) bit in the SSSTAT status register goes High when a transmit or receive operation is in progress. The DE bits control the state of SDEN and enable transmission or reception. A write to the SSTXD0 or SSTXD1 transmit register or a read of the SSRXD receive register initiates the transfer. For a complete functional description of these registers, including the features not used here, refer to the *Am186™CC Communications Controller Register Set Manual*, order #21916.

Assuming the connections in Figure 2 on page 2, the following steps are required to execute the Read Revision Code Number Command (#30 = 73h) of the QSLAC device.

Initialize the SSI registers:

1. Write the PIO Mode 0 register to enable the SSI function on the multiplexed pins [PIOMODE0 bits 10, 11, 12 = 0]
2. Write the PIO Direction 0 register to enable the SSI function on the multiplexed pins [PIODIR0 bits 10, 11, 12 = 0]
3. Write the SSI Mode/Status register to enable SSI enhancements [SSSTAT bit 15 = 1]
4. Write SSI Control register to set MSB first, Low true SDEN, and ÷ 16 [SSCON = 0130h]

Send the command:

1. Enable transmit by setting DE High [SSCON bit 0 = 1]
2. Write the command [SSTXD0 = 73h]
3. Wait for PB to go Low [SSSTAT bit 0 = 0]
4. Disable transmit by setting DE Low [SSCON bit 0 = 0]

After waiting at least 2.5 µs, receive the data:

1. Enable receive by setting DE0 High [SSCON bit 0 = 1]
2. Start reception (read SSRXD)
3. Wait for PB to go Low [SSSTAT bit 0 = 0]
4. Disable receive by setting DE0 Low [SSCON bit 0 = 0]
5. Read revision number (read SSRXD)

Appendix A gives the listings for two general-purpose read and write routines. They have been coded in assembly language to maximize speed. In most cases, the natural flow of the software guarantees that there is at least 5 μs between bytes. In the listing given, the print commands between writes and reads take much longer than 5 μs. If this is not the case, either a software delay or the Am186CC controller's timer could provide the necessary wait.

INITIALIZATION

The initialization process requires two steps. First, the on-board peripheral of the Am186CC controller (PIO and SSI ports) must be set up for proper operation. This includes setting the mode and direction of the PIO signals as well as setting the pin itself to a known state. Second, now that the interface is operational, the SLAC device itself should be initialized. Each of the SLAC devices has a recommended power-up sequence that can be found in the data sheet. For example, the QSLAC device's recommended sequence is as follows:

1. Select MCLK (command #6)
2. Reset software (command #2)
3. Program coefficients and parameters
4. Activate (command #5)

The Am186CC controller's PIO and SSI ports should be initialized as soon as possible after reset to ensure that the output pins are in the correct state. However, after power is stable, 1 ms is needed before commands can be sent to the SLAC device. Most systems have an external power-up-reset monitor that provides this delay. If not, or if they have separate power supplies, software must wait before sending the first command. The QSLAC device has a power interruption flag (PI, command 23, bit 7), which should be checked after the delay.

SOFTWARE LISTING

The software listing given in Appendix A is written in C and compiled with Microsoft's C/C++ compiler. This example code illustrates how to read the QSLAC device's Z-filter coefficients. The software was tested on several of the evaluation boards available from AMD. An ASLAC™ Device Interface Board (ACIF™ board) was used to load a known set of coefficients into a QSLAC device low-noise board. An Am186CC controller demonstration board was then connected in place of the ACIF board to read back the coefficients.

The main body of the program first initializes the various ports and then sends a read Z-filter command. The for loop then reads back the 15 bytes of the Z-filter coefficients. The subroutines, SLAC_read and SLAC_write, are written in assembly language for speed and clarity. These subroutines implement the code required to send or receive a single byte.

Appendix B shows how to modify the write routine to use PIOs instead of SDEN. This example makes use of the Am186CC controller's new PIO set and clear registers to ensure that no other PIOs are affected by this low-level routine. The new registers make it easy to set the state of a specific PIO without affecting any other PIOs, even though all 16 bits are written. The modification of the other routines is left to the reader.

SUMMARY

The new features of the Am186CC controller make it easy to interface with AMD SLAC devices using the SSI. The telecommunications features of the Am186CC controller make it an excellent choice for higher performance linecards.

Interface Software (Using SDEN)

```
#include <stdio.h>
#include "sys\types.h"
#include "serrano.h"          // defines register addresses

// function prototypes

   void  SLAC_init(void);
   void  SLAC_write(int);
   Uint8 SLAC_read(void);

// useful constants

   #define   READ_Z    0x85     // read z-filter
   #define   DE_LOW    0xFFFE   // DE bit low bit mask
   #define   DE_HI     0x0001   // DE bit high bit mask
   #define   PB_HI     0x0001   // PB bit high bit mask

void main() {

   Uint8   buf[256];
   int     i;

   printf("Start Program\n");

   SLAC_init();
   printf("Finish SLAC_init\n");

   for (i=0; i<256; i++) buf[i] = 0;
   printf("Finish buffer initialization\n");

   SLAC_write(READ_Z);
   printf("Command Sent\n");

   for (i=0; i<15; i++) {

       buf[i] = SLAC_read();

       printf("byte %d = %x \n",i,buf[i]);

       }  /* end for loop */

exit(0);
}

//****************************** END OF MAIN ******************************
void SLAC_init(void)
{

   _asm{
```

```
    mov  dx,PIOMODE0  // point to mode register
    in   ax,dx        // read register (in IO space)
    and  ax,0xE3FF    // set bits low (to turn on SSI)
    out  dx,ax        // write to register (in IO space)

    mov  dx,PIODIR0     // point to PIO1 DIRECTION register

    in   ax,dx        // read
    and  ax,0xE3FF    // set bit low (to turn on SSI)
    out  dx,ax        // write

    mov  dx,SSSTAT    // point to sync serial control register
    mov  ax,0x8000    // set ENHCTL (turn on enhancements)
    out  dx,ax        // write

    mov  dx,SSCON     // point to sync serial control register
    mov  ax,0x0130    // set clk divisor to 16, MSB first
    out  dx,ax        // write

    } /* end _asm */


}
// Bit Settings for SSCON:
//
//  CLKP   = 0     ;low active clock
//  DENP   = 0     ;low active SDEN0
//  MSBF   = 1     ;send msb first
//  CLKEXP = 011   ;divide processor clock by 16
//  DE1    = 0     ;don't start transmission yet
//  DE0    = 0     ;ditto
//
//============================ END OF SLAC_INIT ============================
Uint8 SLAC_read(void)
{

   int i;
   _asm{

    mov  dx,SSCON     // STEP 1 - enable reception
    in   ax,dx        //          (i.e. set bit DE0 = 1)
    or   ax,DE0_HI    //
    out  dx,ax        //

    mov  dx,SSRXD     // STEP 2 - start reception
    in   ax,dx        //          (with dummy read of SSR)

    mov  dx,SSSTAT    // STEP 3 - wait for data
h1: in   ax,dx        //          (done when PB = 0)
    and  ax,PB_HI     //
    jnz  h1           //

    mov  dx,SSCON     // STEP 4 - disable reception
    in   ax,dx        //          (set DE0 low)
    and  ax,DE0_LOW   //
    out  dx,ax        //
```

```
    mov  dx,SSRXD     // STEP 5 - read the data
    in   ax,dx        //

    mov  i,ax         // move data to output variable
    } /* end _asm */

    return(i);
}
//============================ END OF SLAC_READ ============================
static void SLAC_write(i)
int i;
{

   _asm{

    mov  dx,SSCON     // STEP 1 - enable transmission
    in   ax,dx        //          (i.e. set bit DE0 = 1)
    or   ax,DE0_HI    //
    out  dx,ax        //

    mov  dx,SSTXD0    // STEP 2 - transmit byte
    mov  ax,i         //
    out  dx,ax        //

    mov  dx,SSSTAT    // STEP 3 - wait for completion
h1: in   ax,dx        //          (done when PB = 0)
    and  ax,PB_HI     //
    jnz  h1           //

    mov  dx,SSCON     // STEP 4 - disable transmission
    in   ax,dx        //          (set DE0 low)
    and  ax,DE0_LOW   //
    out  dx,ax        //

    } /* end _asm */


}
//========================== END OF SLAC_WRITE ============================
```

Modifying the Write Routine to Use PIO Enable Instead of SDEN

```
static void SSI_write(i)
int i;
{

   _asm{

    mov  dx,PCLR10     // STEP 1 - set CS1 low (PIO 10)

    mov  ax,P10       //          using new PIO clear
    out  dx,ax        //          function.

    mov  dx,SSCON     // STEP 2 - enable transmission
    in   ax,dx        //          (i.e. set bit DE0 = 1)
    or   ax,DE0_HI    //
    out  dx,ax        //

    mov  dx,SSTXD0    // STEP 3 - transmit byte
    mov  ax,i         //
    out  dx,ax        //

    mov  dx,SSSTAT    // STEP 4 - wait for completion
h1: in   ax,dx        //          (done when PB = 0)
    and  ax,PB_HI     //
    jnz  h1           //
    mov  dx,SSCON     // STEP 5 - disable transmission
    in   ax,dx        //          (set DE0 low)
    and  ax,DE0_LOW   //
    out  dx,ax        //

    mov  dx,PSET10     // STEP 6 - set CS1 high (PIO 10)

    mov  ax,P10       //          P10 = 0000010000000000b
    out  dx,ax        //

    } /* end _asm */



}
//========================== END OF SSI_WRITE ==============================
```

Trademarks